グラフの構造情報を用いた ObjectRank の高速化

佐藤　朋紀† 　塩川　浩昭†† 　北川　博之††

† 筑波大学大学院システム情報工学研究科　〒 305–8573 茨城県つくば市天王台 1–1–1
†† 筑波大学計算科学研究センター　〒 305–8577 茨城県つくば市天王台 1–1–1
E-mail: †t.sato@kde.cs.tsukuba.ac.jp, ††{shiokawa,kitagawa}@cs.tsukuba.ac.jp

あらまし　ObjectRank は，データベース内のオブジェクトをグラフのノードとみなすことでキーワード検索を実現するグラフ分析手法である．しかしながら，ObjectRank は重要度評価の際にノード数に等しいサイズの行列ベクトル積を繰り返し計算する必要があるため，大規模なグラフへの適用が難しいという問題がある．そこで本稿では，ObjectRank の Top-k 検索の高速化手法を提案する．提案手法は，繰り返し計算の過程で各ノードの重要度の上限値と下限値を計算することで，解になり得ないノードをグラフから逐次的に枝刈りする．このとき，提案手法はグラフの構造情報を用いることで，枝刈りのさらなる高速化を図る．本稿では実データを用いた評価実験を行い，ObjectRank に対する提案手法の有効性を検証する．

キーワード　ObjectRank，グラフマイニング

1 序　　論

ソーシャルメディアやモバイル機器の普及に伴い日々大量のデータが生成されており，データから新たに知識を獲得するデータ分析の需要が増加している．特に，データ間の関係性を表現するグラフと呼ばれるデータ構造に着目したグラフ分析は，データ間の関係性に隠れた知識を獲得できる分析技術であり，推薦システムや SNS(Social Networking Service) といったサービスで幅広く用いられている [1], [9], [17].

グラフ分析技術のひとつに，ObjectRank [2], [10] がある．ObjectRank は，PageRank [3] を拡張することでデータベース内のオブジェクトに対するキーワード検索を実現する手法である．ObjectRank はデータベース内のオブジェクトをグラフに見立てることによってランダムウォークに基づくリンク解析手法を適用し，各オブジェクトの重要度を評価しランク付けする．PageRank とは異なり，ObjectRank は複数の種類のノードとエッジからなるグラフを扱うことができるため，多様なデータに対して適用可能である [5], [8], [11], [14], [24].

ObjectRank は多様なアプリケーションに対して適用可能である一方で，計算コストが膨大であるという問題がある．ObjectRank はクエリが与えられると全てのノードの重要度が収束するまで反復計算を行う必要があるため，大規模なグラフを用いた際に実用的な時間でクエリ応答することは困難である．また，実世界における ObjectRank のアプリケーションでは，上位 k 件のノードのランキングのみが必要となる場合が大多数であるが，従来の解法では全てのノードの重要度を導出して初めて上位 k 件のノードが得られる．一般的に k の値は多くとも数百程度であるのに対し，近年では数億を超えるノード数をもつグラフも存在するため，下位 $N-k$ 件のノードの重要度評価が大きなボトルネックとなってしまう．したがって，ObjectRank の高速化は重要な研究課題となっている．

ObjectRank の重要度評価の高速化手法として，様々な手法が提案されてきた [4], [15], [16]．これらの手法は，グラフから一部のノードを取り除くことで計算対象のノード数を削減したり，効率的なクエリ応答のために索引を事前計算することで高速な計算を実現した．しかしながら，(1) 事前計算に膨大な時間を要する，(2) 大規模なグラフを対象とした場合に依然として計算時間を要するといった問題が存在する．

1.1 本研究の貢献

本研究では，ObjectRank の Top-k 検索を高速化する手法を提案する．提案手法では，繰返し計算の各イテレーションにおいて重要度の上限値と下限値を推定し，Top-k 検索の結果になり得ないノードをグラフから逐次的に枝刈りする．この手法により重要度評価の繰返し計算を進めるにつれて計算対象のノードが減少するため，従来の ObjectRank と比較して Top-k 検索を高速に行うことができる．提案手法はグラフからノードの枝刈りを行うが，得られる上位 k 件のノードは従来の ObjectRank により得られる結果と一致することが保証される．

また，我々が行った予備実験により，提案手法は繰返し計算の序盤では枝刈りの上限を満たすノードが極めて少なく，ノード全体と比較して約 1 % のノードに対してのみ枝刈りが行われ，繰返し計算の終盤の数回で残りの大部分のノードが枝刈りされることが明らかとなった．そこで提案手法は，グラフの構造情報に基づいてグラフの各タイプごとの重要度の分布を事前に導出し，よりタイトな上限値を設定することでさらなる高速化を図る．これにより，提案手法は繰返し計算の序盤においてもより多くのノードの枝刈りが可能となる．

本研究の貢献は下記の通りである．

- **高速性:** 提案手法は従来のべき乗法を用いた ObjectRank と比較して上位 k 件の結果を高速に計算することができる．提案手法は 100 万のノードと 500 万のエッジを持つグラフに対し，約 2 秒で検索結果を得ることができる（ 4.1 節）．

表 1　本稿が用いる記号の定義

記号	定義
G	計算対象のグラフ
V	G に含まれるノードの集合
E	G に含まれるエッジの集合
N	グラフのノード数
M	グラフのエッジ数
$\boldsymbol{A}$	$N \times N$ の遷移行列
d	ダンピングファクタ
$\boldsymbol{r}_t$	キーワード t に対する各ノードの重要度を並べた $N \times 1$ のベクトル
$\boldsymbol{q}_t$	キーワード t についてのクエリベクトル
$BS(t)$	キーワード t をもつノードの集合

- **正確性:** 提案手法は ObjectRank の上位 k 件の結果を正確に導出することができる（ 4.2 節）.

上述の通り，ObjectRank はすべてのノードの重要度が収束するまで反復計算を行う必要があるため，大規模グラフに適用することは難しい．しかし，提案手法は逐次的なノードの枝刈りを行うことで計算対象のノード数を削減し，上位 k 件の結果を高速に得ることができる．実データを用いた評価実験では，従来のべき乗法を用いた ObjectRank と比較して約 5 倍高速に計算できることを示した．

本稿の構成は次の通りである．まず， 2 節で前提となる知識について説明し， 3 節で提案手法について述べる． 4 節で評価実験について説明し， 5 節で関連研究を紹介する．そして最後に 6 節でまとめを述べる．

2　前提知識：ObjectRank

本章では，本研究に関する基本事項について説明する．表 1 に本稿が用いる記号とその定義を示す．以降では，2.1 節で ObjectRank の概要について，2.1.1 節でグラフの構築方法について，2.1.2 節で重要度評価の方法についてそれぞれ述べる．

2.1　ObjectRank の概要

ObjectRank [2], [10] は，PageRank [3] を拡張することで複数種類のノードとエッジからなるグラフに対するキーワード検索を実現する手法である．具体的には，まずはじめに分析対象のデータをグラフを用いて表現し，その後ユーザからクエリとして与えられたキーワードに対する各ノードの重要度を計算しランキングを導出する．グラフを構築する際に，あらかじめ分析対象のデータベースに基づいてグラフを構成するノードとエッジのタイプを定義し，エッジのタイプごとに重みを割り当てる．このとき，各エッジにはタイプに応じて異なる重みを付与することが可能であるため，重要度評価の際にノード間の関係の差異を十分に捉えることができる．それに加えて，ObjectRank はクエリで与えられたキーワードに基づいて一部のノードの集合に対して重み付けを行うことによって，クエリに応じた検索結果を得ることができる．

2.1.1　グラフの構築方法

ObjectRank はデータベース内のオブジェクトをラベル付き有向グラフとして表現する．まず，グラフのノードとエッジの種類，及びエッジの重みを定義した *Authority Transfer Schema Graph* を作成する（以降，*Schema Graph* と呼ぶ）．各種ノードとエッジはラベルが付与され，ラベルによって種類が区別される．各エッジの重みはそのエッジによって遷移する重要度の割合を示す．ただし，重みは 0 以上 1 以下の値をとり，一つのノードから出るエッジの重みの総和は 1 以下に設定する必要がある．図 1 に文献データベースを表現した *Schema Graph* の例を示す．図 1 の例では，“Conference” や “Year” といったラベルが付与された 4 種類のノードと，それらをつなぐ 8 種類のエッジが存在する．

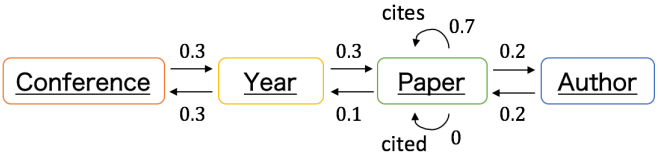

図 1　文献データベースの *Schema Graph*

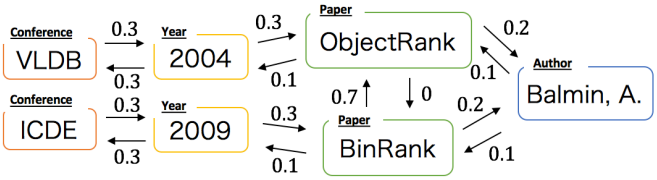

図 2　図 1 に基づいて作成された *Data Graph* の例

次に，*Schema Graph* に基づいて対象の全データを実体化したグラフ *Authority Transfer Data Graph* を構築する（以降，*Data Graph* と呼ぶ）．*Data Graph* の各ノードはデータベース内の 1 つのオブジェクトに対応し，オブジェクトに含まれるキーワード集合を保持する．*Data Graph* の各エッジの重みは，*Schema Graph* で定義した重みをエッジの元ノードの出次数で割った値となる．ただし，出自数は同じ種類のエッジの本数のみを数えたものとする．図 2 に 図 1 に基づいて作成された文献データベースの *Data Graph* の例を示す．図 2 では，“Balmin, A.” ノードから “ObjectRank” ノードには重みが 0.1 であるエッジが張られている．これは，*Schema Graph* の “Author” から “Paper” へのエッジの重みである 0.2 を，“Balmin, A.” の出次数 2 で割った値である．

2.1.2　重要度の計算方法

ObjectRank は，上述の方法により構築された *Data Graph* に対して重要度評価を行う．ノードとエッジの集合をそれぞれ V，E，ノード数とエッジ数をそれぞれ N，M とする．あるキーワード t と *Data Graph* $G = (V, E)$ が与えられたとき，ObjectRank はキーワード t に対する各ノードの重要度を並べたベクトル $\boldsymbol{r}_t = [r_t(v_1), r_t(v_2), \ldots, r_t(v_N)]$ をランダムウォークモデル [3] に基づいて計算する．すなわち，キーワード t を持つノード集合を $BS(t)$ とすると，ランダムサーファは $BS(t)$ に含まれるランダムなノードを始点として (1) エッジの重みに応じた確率で隣接ノードへと遷移，または (2) $BS(t)$ のランダムなノードへとジャンプする行動を繰り返す．ある時点でランダムサーファが滞在している確率を，そのノードの重要度と見なす．これらの処理は以下の式 (1) で定式化することができる．

$$\boldsymbol{r}_t = d\boldsymbol{A}\boldsymbol{r}_t + (1-d)\boldsymbol{q}_t \tag{1}$$

ただし，$\boldsymbol{A}$ は $N \times N$ の遷移行列であり，(i,j) 要素にはノード v_j から v_i にエッジ e_{ji} が存在する場合は e_{ji} の重みを，それ以外の場合は 0 を格納する．d $(0 < d < 1)$ はダンピングファクタ，$\boldsymbol{q}_t$ は n 次元のクエリベクトルであり，ノード v が $v \in BS(t)$ を満たすとき $q_t(v) = 1/|BS(t)|$，それ以外の場合は $q_t(v) = 0$ となる．

ObjectRank は，べき乗法を用いて式 (1) を解く．すなわち，任意の初期値を与えて以下の式を繰り返し計算する．

$$\boldsymbol{r}_t^{(i+1)} = d\boldsymbol{A}\boldsymbol{r}_t^{(i)} + (1-d)\boldsymbol{q}_t \tag{2}$$

$\boldsymbol{r}_t^{(i)}$ は i 回目の繰返し計算によって求めた重要度のベクトルである．$\boldsymbol{r}_t^{(i)}$ が収束したとき，繰返し計算を終了する．

3 提 案 手 法

3.1 提案手法の概要

前章で述べた通り，ObjectRank はノード数に比例したサイズの行列ベクトル積を繰り返し計算する必要があるため，グラフが大規模化した際に計算コストが爆発的に増大するという問題がある．また，実世界における ObjectRank のアプリケーションでは，上位 k 件のノードのランキングのみが必要となる場合が大多数であるが，従来の解法では全てのノードの重要度が収束するまで繰り返し計算をして初めて上位 k 件のノードが得られる．一般的に k の値は多くとも数百程度であるのに対し，近年では数億を超えるノード数をもつグラフも存在するため，下位 $N-k$ 件のノードの重要度評価が大きなボトルネックとなってしまう．

そこで本研究では，重要度評価の際に逐次的にノードを枝刈りすることで ObjectRank の Top-k 検索の高速化を図る．我々は，各イテレーションにおいて重要度の上限値と下限値が導出可能であるという性質を発見した．そこで本研究では，この性質を利用して逐次的に上位 k 件の検索結果になり得ないノードを枝刈りする手法を提案する．提案手法は，グラフの全ノードの重要度を繰り返し計算する従来の手法と比較して計算対象のノード数を著しく削減できるため，高速な処理が可能となる．

また，我々が行った予備実験により，上述の手法は繰り返し計算の序盤では枝刈りの条件を満たすノードが極めて少ないため全体のノードと比較して約 1 % のノードに対してのみ枝刈りが行われ，繰り返し計算の終盤において残りの大部分のノードが枝刈りされることが明らかとなった．そこで提案手法は，上述の上限値と下限値を用いた逐次的な枝刈りを行う際に，グラフの構造情報を利用してさらなる高速化を図る．後述の補題 36 より，Schema Graph は Data Graph のエッジの重みやエッジの接続関係を定義したグラフであり，Schema Graph に対して ObjectRank を実行することでノードのタイプごとの重要度の分布を導出できることが明らかとなった．SchemaRank はこの性質を利用して，Data Graph に対して Top-k 検索を行う前に Schema Graph に対して ObjectRank を実行することでノードのタイプごとの重要度の分布を導出し，逐次的なノードの枝刈りの際に用いることで高速化を図る．

以降では， 3.2 節で重要度の上限値と下限値の性質について述べ， 3.3 節で上限値と下限値を用いた提案手法の基本的なアルゴリズムを説明する．そして 3.4 節で Schema Graph に対する ObjectRank の実行とその性質の議論を行い， 3.5 節でグラフの構造情報を用いた提案手法のアルゴリズムを説明する．

3.2 上限値と下限値

2.1.2 節で説明した通り，ObjectRank の重要度は式 (2) を用いた繰り返し演算によって計算される．つまり，あるノード v について注目したとき，ノード v の重要度は初期値 $r_t^{(0)}(v)$ が各反復において増減し，ある値 $r_t(v)$ へと収束する．上限値 $\overline{r}_t^{(i)}(v)$ と下限値 $\underline{r}_t^{(i)}(v)$ は，この収束値 $r_t(v)$ に対して $\underline{r}_t^{(i)}(v) \leqq r_t(v) \leqq \overline{r}_t^{(i)}(v)$ の性質を満たす値である．

3.2.1 上限値と下限値の定義とその性質

$\boldsymbol{p}_t^{(i)}$ を長さ i のランダムウォークの確率を表す N 次元ベクトルとする．$\boldsymbol{p}_t^{(i)} = \boldsymbol{A}^i\boldsymbol{q}_t$ で計算し，$i = 0$ のとき $\boldsymbol{p}_i = \boldsymbol{e}$ である．ただし，$\boldsymbol{e}$ は各要素が 1 である N 次元ベクトルとする．このとき，i 番目の繰り返し計算における ObjectRank スコアの下限値 $\underline{\boldsymbol{r}}_t^{(i)}$ と上限値 $\overline{\boldsymbol{r}}_t^{(i)}$ を以下に定義する．

定義 31 (下限値)．i 番目の繰り返し計算におけるノード v の下限値は次のように計算する．

$$\underline{r}_t^{(i)}(v) = (1-d)\sum_{j=0}^{i} d^j p_t^{(j)}(v) \tag{3}$$

定義 32 (上限値)．i 番目の繰り返し計算におけるノード v の上限値は次のように計算する．

$$\begin{aligned}\overline{r}_t^{(i)}(v) =& (1-d)\sum_{j=0}^{i} d^j p_t^{(j)}(v) \\ &+ d^{i+1}p_t^{(i)}(v) + \frac{d^{i+1}}{1-d}\Delta_t^{(i)}\overline{A}(v)\end{aligned} \tag{4}$$

また，$\Delta_t^{(i)}$ は次のように計算する．

$$\Delta_t^{(i)} = \begin{cases} 1 & (i=1) \\ \sum_{u \in V_0} \Delta_t^{(i)}(u) & (otherwise) \end{cases}$$

ただし，G_i を i 番目の繰り返し計算における部分グラフ，G_0 を元のグラフとし，V_0 は G_0 のノード集合を表す．$\Delta_t^{(i)}(v)$ は $\Delta_t^{(i)}(v) = max\{p_t^{(i)}(v) - p_t^{(i-1)}(v), 0\}$ により計算する．$\overline{\boldsymbol{A}}$ はエッジの最大の重みを格納した N 次元ベクトルであり，$\overline{A}(v) = max\{A(v,u) : u \in G_i\}$ となる．

補題 31 (下限値の性質)．i 番目の繰り返し計算において，下限値 $\underline{r}_t^{(i)}(v)$ は次の性質を満たす．

$$\underline{r}_t^{(i)}(v) \leqq r_t(v) \tag{5}$$

証明． 式 (2) より，

$$\begin{aligned}\boldsymbol{r}_t^{(i)} &= d\boldsymbol{A}\boldsymbol{r}_t^{(i-1)} + (1-d)\boldsymbol{q}_t \\ &= d^2\boldsymbol{A}^2\boldsymbol{r}_t^{(i-2)} + (1-d)(d\boldsymbol{A}\boldsymbol{q}_t + \boldsymbol{q}_t) \\ &= d^i\boldsymbol{A}^i\boldsymbol{r}_t^{(0)} + (1-d)\{d^{i-1}\boldsymbol{A}^{i-1}\boldsymbol{q}_t + \cdots + \boldsymbol{q}_t\} \\ &= d^i\boldsymbol{A}^i\boldsymbol{r}_t^{(0)} + (1-d)\sum_{j=0}^{i-1} d^j\boldsymbol{p}_t^{(j)}\end{aligned}$$

最終的な ObjectRank のスコアベクトルは $\boldsymbol{r}_t^{(i)}$ の収束値であるため，$\boldsymbol{r}_t = \boldsymbol{r}_t^{(\infty)}$ となる．$0 < d < 1$ かつ $\boldsymbol{A}^\infty$ の各要素は 0 以上 1 以下の値をとるため，

$$\boldsymbol{r}_t = d^\infty\boldsymbol{A}^\infty\boldsymbol{r}_t^{(0)} + (1-d)\sum_{j=0}^{\infty} d^j\boldsymbol{p}_t^{(j)} = (1-d)\sum_{j=0}^{\infty} d^j\boldsymbol{p}_t^{(j)} \tag{6}$$

が成り立つ．したがって，

$$\boldsymbol{r}_t = (1-d)\sum_{j=0}^{\infty} d^j\boldsymbol{p}_t^{(j)} \geqq (1-d)\sum_{j=0}^{i} d^j\boldsymbol{p}_t^{(j)} \tag{7}$$

が成り立つため，補題 31 の式 (5) が成立する． □

補題 32 (上限値の性質)．i 番目の繰り返し計算において，上限値 $\overline{r}_t^{(i)}(v)$ は次の性質を満たす．

$$\overline{r}_t^{(i)}(v) \geqq r_t(v) \tag{8}$$

証明． スペースの都合により省略する． □

また，上限値と下限値は繰り返し計算を進めると最終的に重要度の収束値に一致することが保証される．

補題 33 (上限値と下限値の収束性)．上限値 $\overline{r}_t^{(i)}$ と下限値 $\underline{r}_t^{(i)}$ は最終的に従来の *ObjectRank* によって得られる重要度の収束値 $r_t(v)$ に一致する．すなわち，$\underline{r}_t^{(\infty)}(v) = r_t(v) = \overline{r}_t^{(\infty)}(v)$ を満たす．

証明． 式 (31) より $\underline{r}_t^{(\infty)}(v) = (1-d)\sum_{j=0}^{\infty} d^j p_t^{(j)}(v)$ となるため，明らかに $\underline{r}_t^{(\infty)}(v) = r_t(v)$ が成り立つ．式 (32) より $\overline{r}_t^{(i)}(v) = (1-d)\sum_{j=0}^{\infty} d^j p_t^{(j)}(v) + d^\infty p_t^{(\infty)}(v) + \frac{d^\infty}{1-d}\Delta_t^{(\infty)}\overline{A}(v)$ となる．ここで，$0 < d < 1$ かつ $0 \leqq p_t^{(\infty)}(v) \leqq 1$ であるため，$d^\infty p_t^{(\infty)}(v) = 0$ となる．同様に，$0 \leqq \Delta_t^{(\infty)} \leqq 1$ かつ $0 \leqq \overline{A}(v) \leqq 1$ であるため，$\frac{d^\infty}{1-d}\Delta_t^{(\infty)}\overline{A}(v) = 0$ とある．ゆえに，$\overline{r}_t^{(\infty)}(v) = (1-d)\sum_{j=0}^{\infty} d^j p_t^{(j)}(v)$ となるため，$r_t(v) = \overline{r}_t^{(\infty)}(v)$ が成り立つ．したがって，$\underline{r}_t^{(\infty)}(v) = r_t(v) = \overline{r}_t^{(\infty)}(v)$ が成り立つ． □

重要度評価の際の繰り返し計算において，下限値 $\underline{\boldsymbol{r}}_t^{(i)}$ と上限値 $\overline{\boldsymbol{r}}_t^{(i)}$ は次のように逐次的に計算することができる．

補題 34 (下限値と上限値の逐次的な計算)．i 番目の繰り返し計算において，下限値 $\underline{r}_t^{(i)}(v)$ と上限値 $\overline{r}_t^{(i)}(v)$ は次式で計算する．また，次式は $p_t^{(i)}(v)$ が計算済みであるとき $O(1)$ で計算可能である．

$$\underline{r}_t^{(i)}(v) = \begin{cases} (1-d)q_t(v) & (i=0) \\ \underline{r}_t^{(i-1)} + (1-d)d^i p_t^{(i)}(v) & (i \neq 0) \end{cases} \tag{9}$$

Algorithm 1 Our basic proposed algorithm

Input: G, given graph; t, keyword; k, # of answer vertices
Output: Top-k vertices from final subgraph G_{i+1}

```
1: i ← 0
2: G_i ← G
3: repeat
4:     for each v ∈ G_i do
5:         calculate r_t^(i)(v) and r̄_t^(i)(v) by equation (9) and (10)
6:     end for
7:     ε_i ← the k-th largest lower bound in G_i
8:     construct G_{i+1} by pruning vertices s.t {v ∈ V_i : r̄_t^(i)(v) < ε_i} from G_i
9:     i ← i + 1
10: until |V_{i+1}| = k
11: return V_{i+1}
```

$$\overline{r}_t^{(i)}(v) = \begin{cases} q_t(v) + \frac{d}{1-d}\overline{A}(v) & (i=0) \\ \underline{r}_t^{(i-1)} + d^i p_t^{(i)}(v) + \frac{d^{i+1}}{1-d}\Delta_t^{(i)}\overline{A}(v) & (i \neq 0) \end{cases} \tag{10}$$

証明． $i = 0$ のとき，定義 31 より $\underline{r}_t^{(i)}(v) = (1-d)\sum_{j=0}^{i} d^j p_t^{(j)}(v) = (1-d)p_t^{(0)}(v) = (1-d)q_t(v)$ が成り立つ．また，定義 32 より $\overline{r}_t^{(i)}(v) = (1-d)q_t(v) + dq_t(v) + \frac{d}{(1-d)}\overline{A}(v) = q_t(v) + \frac{d}{(1-d)}\overline{A}(v)$ が成り立つ．このとき，d，$q_t(v)$，$\overline{A}(v)$ は定数であるため，式 (9)，(10) は $O(1)$ で計算可能である．

次に，$i \neq 0$ のとき，定義 31 より $\underline{r}_t^{(i)}(v) - \underline{r}_t^{(i-1)}(v) = (1-d)d^i p_t^{(i)}(v)$ となり，$\underline{r}_t^{(i)}(v) = \underline{r}_t^{(i-1)} + (1-d)d^i p_t^{(i)}(v)$ が成り立つ．また，定義 31，32 より $\overline{r}_t^{(i)}(v) - \underline{r}_t^{(i-1)}(v) = d^i p_t^{(i)}(v) + \frac{d^{i+1}}{(1-d)}\Delta_t^{(i)}\overline{A}(v)$ となり，$\overline{r}_t^{(i)}(v) = \underline{r}_t^{(i-1)}(v) + d^i p_t^{(i)}(v) + \frac{d^{i+1}}{(1-d)}\Delta_t^{(i)}\overline{A}(v)$ が成り立つ．$\underline{r}_t^{(i-1)}(v)$ はあらかじめ計算されており，d，$p_t^{(i)}(v)$，$\overline{A}(v)$ は定数であるため，式 (9)，(10) は $O(1)$ で計算できる． □

3.3 基本的なアルゴリズム

上限値と下限値を用いた提案手法の基本的なアルゴリズムを Algorithm 1 に示す．i $(i = 0, 1, \ldots)$ 番目の繰り返し計算において，各ノードの重要度の上限値 $\overline{r}_t^{(i)}(v)$，下限値 $\underline{r}_t^{(i)}(v)$ を計算する（5 行目）．閾値 ϵ_i を i 番目の繰り返し計算における k 番目に大きい下限値とする（7 行目）．このとき，上限値が ϵ_i を下回ったノードをグラフ G_i から取り除き，新たな部分グラフ G_{i+1} を構築する（8 行目）．G_{i+1} に含まれるノード数 $|V_{i+1}|$ が k に達したとき計算を終了し，k 件のノードを返す．

Algorithm 1 が従来の ObjectRank と同様の結果が得られることを示すために，以下の補題を導入する．

補題 35 (アルゴリズムの正確性)．提案手法は，従来の *ObjectRank* によって得られる上位 k 件のノードを正確に検索することができる．

証明． 従来の ObjectRank によって計算される重要度の収束値の k 番目に大きな値を ϵ とする．すなわち，あるノード v が上位 k 件の検索結果に含まれるとき，重要度の収束値 $r_t(v)$ は必ず $\epsilon \leqq r_t(v)$ を満たす．i 番目の繰り返し計算における k 番目に大きな下限値を ϵ_i とすると，補題 31 より $\epsilon_i \leqq \epsilon$ が成り立つ．提案手法は $\overline{r}_t^{(i)}(u) < \epsilon_i$ を満たすようなノード u をグラフから枝刈りするが，補題 32 よりノード u について

Algorithm 2 ObjectRank on Schema Graph

Input: G_S, Schema Graph; t, keyword; τ, maximum # of iterations
Output: ObjectRank scores of all vertices in V_S

```
Initialize r_{S,t}^{(0)} by Equation (11)
Set query vector q_{S,t} by Equation (12)
i ← 0
while i < τ do
    Calculate r_{S,t}^{(i)} by Equation (13)
    if All of ObjectRank scores converges then
        Stop algorithm
    end if
    i ← i + 1
end while
```

$r_t(u) \leqq \overline{r}_t^{(i)}(u) < \epsilon$ が成り立つ．これは検索結果に含まれる場合の条件 $\epsilon \leqq r_t(v)$ に矛盾するため，提案手法は上位 k 件になり得ないノードを必ず枝刈りする． □

3.4 Schema Graph に対する ObjectRank

Algorithm 2 に Schema Graph $G_S = (V_S, E_S)$ に対して ObjectRank を実行する際のアルゴリズムを示す．Schema Graph に対する ObjectRank は (1) 重要度のベクトル $\boldsymbol{r}_{S,t}$ の初期値 $\boldsymbol{r}_{S,t}^{(0)}$ の設定方法および (2) クエリベクトル $\boldsymbol{q}_{S,t}$ の作成方法を除き，Data Graph に対する通常の ObjectRank と同様に行う．

重要度の初期値: Schema Graph のノードタイプ λ_i の集合を $\Lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_{N_S}\}$ とする．ただし，N_S はノードのタイプの総数，つまり，Schema Graph のノード数である．このとき，タイプが λ_i である Schema Graph のノード v_{λ_i} の重要度の初期値は

$$r_{S,t}^{(0)}(v_{\lambda_i}) = \frac{N_{\lambda_i}}{N} \tag{11}$$

とする．ただし，N および N_{λ_i} はそれぞれ Data Graph のノード数，タイプが λ_i である Data Graph のノードの総数である．

クエリベクトルの作成: タイプが λ_i である Schema Graph のノード v_{λ_i} について，クエリベクトルの値は

$$q_{S,t}(v_{\lambda_i}) = \frac{|BS_{\lambda_i}(t)|}{|BS(t)|} \tag{12}$$

とする．ただし，$|BS_{\lambda_i}(t)|$ はタイプが λ_i である Data Graph のノードのうち，キーワード t をもつノードの総数を表す．

上述の通り重要度の初期値とクエリベクトルを作成したとき，Schema Graph の各ノードの重要度 $\boldsymbol{r}_{S,t} = (r_{S,t}(v_1), r_{S,t}(v_2), \ldots, r_{S,t}(v_{N_S}))$ は以下の式 (13) を重要度のベクトル $\boldsymbol{r}_{S,t}$ が収束するまで繰り返し計算することで得られる．

$$\boldsymbol{r}_{S,t}^{(i)} = d\boldsymbol{A}_S\boldsymbol{r}_{S,t}^{(i-1)} + (1-d)\boldsymbol{q}_{S,t}. \tag{13}$$

このとき，$\boldsymbol{A}_S$ は Schema Graph G_S の遷移行列であり，(i,j) 要素にはノード v_j $(\in V_S)$ から v_i $(\in V_S)$ にエッジ e_{ji} $(\in E_S)$ が存在する場合は e_{ji} の重み $\alpha(e_{ij})$ を，それ以外の場合は 0 を格納する．

3.4.1 Schema Graph と Data Graph の重要度の分布

Algorithm 2 によって得られた重要度 $\boldsymbol{r}_{S,t}$ について，以下の補題が成り立つ．

補題 36 (Schema Graph と Data Graph の重要度の分布)**.** *Algorithm 2* によって得られた重要度 $\boldsymbol{r}_{S,t}$ と，この *Schema Graph* から作成された *Data Graph* に *ObjectRank* を実行して得られる重要度 $\boldsymbol{r}_t$ には，任意のタイプ $\lambda_i \in \Lambda$ について以下の関係が成り立つ．

$$r_{S,t}(v_{\lambda_i}) = \sum_{v \in V(\lambda_i)} r_t(v). \tag{14}$$

ただし，$V(\lambda_i)$ はタイプが λ_i である *Data Graph* のノードの集合である．

証明. 3.4 節で述べた通り，$r_{S,t}(v_{\lambda_i})$ の初期値は $r_{S,t}^{(0)}(v_{\lambda_i}) = N_{\lambda_i}/N$ である．このとき，Data Graph の任意のノード $v \in V$ の初期値 $r_t^{(0)}(v)$ は $1/N$ であるため

$$r_{S,t}^{(0)}(v_{\lambda_i}) = N_{\lambda_i} \times \frac{1}{N} = \sum_{v \in V(\lambda_i)} r_t^{(0)}(v) \tag{15}$$

が成り立ち，$\boldsymbol{r}_{S,t}$ の初期値と $\boldsymbol{r}_t$ の初期値について，式 (14) が成り立つ．

以降のイテレーションでは (1) 隣接ノード間での重要度の遷移および (2) ランダムジャンプによる重要度の遷移によって各ノードの重要度が増減するが，各イテレーションにおいて必ず式 (14) が成り立つことを証明する．

Data Graph の任意のノード $v \in V$ について，v の隣接ノードの集合を $Out(v)$，$Out(v)$ においてタイプが λ_i であるノードの集合を $Out_{\lambda_i}(v)$ とする．Schema Graph において，タイプ $\lambda(v)$ のノードがエッジを張っているノードのタイプの集合を Λ_v とすると，$Out(v) = \bigcup_{\lambda_i \in \Lambda_v} Out_{\lambda_i}(v)$ が成り立つ．Λ_v の各タイプ $\lambda_i \in \Lambda_v$ について，v から $Out_{\lambda_i}(v)$ のノードへと張られるエッジの重みの総和は，Schema Graph における $\lambda(v)$ から λ_i へのエッジの重みに一致する．つまり，v から $Out_{\lambda_i}(v)$ の各ノードへと重要度が遷移する確率の総和を α とすると，Schema Graph において $\lambda(v)$ から λ_i へと重要度が遷移する確率は α に一致する．また，各イテレーションおいて Data Graph と Schema Graph のタイプごとの重要度の総和が一致しているため，$\sum_{v \in V(\lambda(v))} r_t(v) = r_{S,t}(\lambda(v))$ が成り立つ．したがって，

$$\sum_{v \in V(\lambda(v))} \alpha \times r_t(v) = \alpha \sum_{v \in Out_{\lambda(v)}} r_t(v) = \alpha \times r_{S,t}(\lambda(v))$$

が成り立つ．すなわち，ある 2 つのタイプ λ_1, $\lambda_2 (\in \Lambda)$ について，Data Graph の $V(\lambda_1)$ のノードから $V(\lambda_2)$ のノードへと遷移する重要度の総和は，Schema Graph においてノード v_{λ_1} から v_{λ_2} へと遷移する重要度の値は一致する．

式 (12) より，Schema Graph に対する ObjectRank のクエリベクトルは，$q_{S,t}(v_{\lambda_i}) = \frac{|BS_{\lambda_i}(t)|}{|BS(t)|}$ である．ただし，$|BS_{\lambda_i}(t)|$ はタイプが λ_i である Data Graph のノードのうち，キーワード t をもつノードの総数である．Data Graph の任意の

Algorithm 3 Proposed algorithm using schema information

Input: G, Data Graph; G_S, Schema Graph; t, keyword; k, # of answer vertices
Output: Top-k vertices from final subgraph G_{i+1}
1: // Running ObjectRank on Schema Graph G_S
2: Run Algorithm 2 to get $\boldsymbol{r}_{S,t}$
3:
4: $i \leftarrow 0$
5: $G_i \leftarrow G$
6: **repeat**
7: **for each** $v \in G_i$ **do**
8: Calculate $\underline{r}_t^{(i)}(v)$ and $\overline{r}_t^{(i)}(v)$ by Equation (9) and (10)
9: **end for**
10: $\epsilon_i \leftarrow$ the k-th largest lower bound in G_i
11: Construct G_{i+1} by pruning vertices *s.t.* $\{v \in V_i : \overline{r}_t^{(i)}(v) < \epsilon_i\}$ from G_i
12: $i \leftarrow i + 1$
13: **until** $|V_{i+1}| = k$
14: **return** V_{i+1}

ノード v について，ノード v が $v \in BS(t)$ を満たすとき $q_t(v) = 1/|BS(t)|$，それ以外の場合は $q_t(v) = 0$ であるため，Λ の各タイプ $\lambda_i \in \Lambda$ について

$$\sum_{v \in BS_{\lambda_i}(t)} q_t(v) = |BS_{\lambda_i}(t)| \times \frac{1}{|BS(t)|} = \frac{|BS_{\lambda_i}(t)|}{|BS(t)|} = q_{S,t}(v_{\lambda_i})$$

が成り立つ．したがって，各イテレーションにおいて Data Graph のタイプごとのクエリベクトルの値の総和は，Schema Graph のクエリベクトルの対応するタイプの値に一致する．

以上より，各イテレーションにおいて式 (14) が成り立つため，補題 36 は成り立つ． □

補題 36 より，Data Graph の重要度 $\boldsymbol{r}_t$ について，以下の性質が成り立つ．

補題 37 (Schema Graph と Data Graph の重要度)．*Data Graph* の任意のノード $v \in V$ について，v の重要度 $r_t(v)$ は，タイプが $\lambda(v)$ である *Schema Graph* のノードの重要度 $r_{S,t}(\lambda(v))$ 以下の値をとる．したがって，以下の式が成り立つ．

$$r_t(v) \leqq r_{S,t}(\lambda(v)) \tag{16}$$

証明． 補題 36 より，$\sum_{v \in V(\lambda(v))} r_t(v) = r_{S,t}(\lambda(v))$ であるため，

$$r_t(v) = r_{S,t}(\lambda(v)) - \sum_{u \in V(\lambda(v)) \wedge u \neq v} r_t(u)$$

が成り立つ．このとき，$r_t(u)$ は 0 以上 1 以下の値をとるため，式 (17) は成り立つ． □

3.5 グラフの構造情報を用いた提案手法のアルゴリズム

補題 36，38 に示した性質より，Data Graph のノード v のタイプ $\lambda(v)$ に応じて Schema Graph の重要度 $r_{S,t}(\lambda(v))$ を用いることで，よりタイトな上限値を設定することが可能となる．提案手法は，この性質を利用することにより Alrorithm 1 で示した基本的なアルゴリズムの繰り返し計算の序盤で枝刈りが行われにくいというボトルネックを解消する．

提案手法では，上限値に用いられる長さ i の遷移確率のベクトル $\boldsymbol{p}_t^{(i)}$ の初期値を Schema Graph の重要度 $r_{S,t}(\lambda(v))$ に変更する．$0 \leqq \boldsymbol{p}_t^{(i)} \leqq 1$ かつ $\boldsymbol{p}_t^{(i)}$ は単調減少するため，以下の補題が成り立つ．

補題 38 (グラフの構造情報を用いた重要度の上限値)．ノード v のグラフの構造情報を用いた上限値 $\overline{r}_{S,t}^{(i)}(v)$ は，グラフの構造情報を用いない上限値 $\overline{r}_t^{(i)}(v)$ に対して以下の式が成り立つ．

$$\overline{r}_{S,t}^{(i)}(v) \leqq \overline{r}_t^{(i)}(v) \tag{17}$$

証明． スペースの都合により省略する． □

Algorithm 3 にグラフの構造情報を用いた提案手法のアルゴリズムを示す．提案手法はまずはじめに Alrorithm 2 を呼び出し，Schema Graph の重要度のベクトル $\boldsymbol{r}_{S,t}$ を取得する（2 行目）．Schema Graph のノード数はたかだか数ノード程度であるので，非常に高速に動作する．以降の流れは Algorithm 1 と同様であるが，上限値の計算には Schema Graph の重要度 $\boldsymbol{r}_{S,t}$ を用いる (8 行目)．上位 k 件のノードが得られたとき，計算を終了する．

4 評 価 実 験

本節では，提案手法，従来の ObjectRank および既存手法の BinRank [15] を実行し，効率性と検索結果上位 k 件の近似精度の観点で提案手法の有効性を検証する．

- **提案手法:** Algorithm 1 に示した上限値と下限値を用いた提案手法の基本的なアルゴリズム，および Algorithm 3 に示したグラフの構造情報を用いた提案手法．簡単のため，図中ではそれぞれ “Basic”，“Schema” と表記する．
- **ObjectRank:** べき乗法を用いた従来の計算方法 [2].
- **BinRank:** 全体のグラフと比較してノード数の少ない部分グラフを構築することで高速化を図る手法 [15]．BinRank は，データベースに含まれるキーワードをクラスタリングし，クラスタに含まれるキーワードとの関連度が著しく低いノードを取り除いた部分グラフを，クラスタごとに構築し索引に格納する．クエリ処理時には，クエリキーワードを含むクラスタから作成された部分グラフのみを対象に，べき乗法による重要度評価を行う．

データセットは以下の 2 種類のグラフを用いた．

- **MOVIE:** Wikipedia に記載された情報を構造化したデータベースである YAGO [18] から，映画とその主役，監督に関する情報を抽出したサブセット [12], [13]．ノード数とエッジ数はそれぞれ 173,158，799,322 である．
- **DBLP:** AMiner (`http://aminer.org`) が公開している，論文とその著者，発表年，学会といった文献に関するデータベース [23], [20], [19]．ノード数とエッジ数はそれぞれ 1,238,266，5,149,294 である．

MOVIE の Schema Graph は図 7 に示したものを使用する．DBLP の Schema Graph は図 1 に示した文献 [2] と同様のものを使用した．ダンピングファクタ d は文献 [2] と同様に $d = 0.85$ とした．プログラムは C++ で実装し，計算機は 3.5GHz CPU，メモリが 128GB の Linux サーバを用いた．

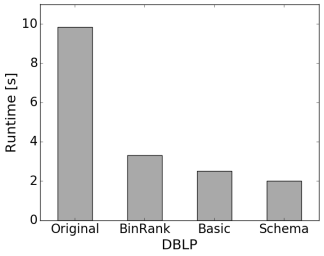

図 3　実行時間（DBLP）

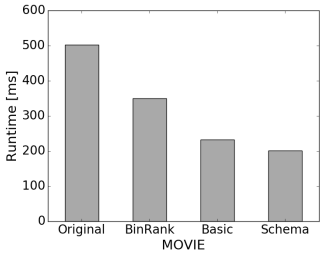

図 4　実行時間（MOVIE）

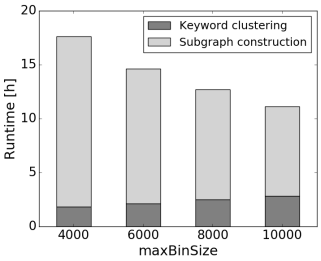

図 5　BinRank の事前計算（DBLP）

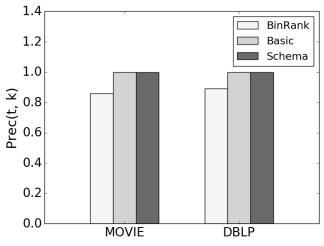

図 6　近似精度

図 7　YAGO データセットの Schema Graph

4.1　効　率　性

上述の 4 つの手法をそれぞれ実行した際の実行時間を評価する．ただし，提案手法は Algorithm 1 に示した上限値下限値を用いた基本的なアルゴリズム（“Basic” と表記する）と Algorithm 3 に示したグラフの構造情報を用いたアルゴリズム（“Schema” と表記する）をそれぞれ実行した．図 3 と図 4 に，DBLP と MOVIE データセットに対して実行した際の実行時間をそれぞれ示す．実験結果より，逐次枝刈りによる提案手法（Basic）は従来のべき乗法を用いた ObjectRank と比較して，約 5 倍程度高速であることが示された．ObjectRank はすべてのノードの重要度が収束するまで繰り返し計算を行う必要があるが，提案手法は解になり得ないノードを逐次的に枝刈りし上位 k 件のノードが得られた時点で計算を終了するため，高速な計算が可能となる．グラフの構造情報を用いた提案手法（Schema）は，全ての手法の中で最も高速であることが示された．この結果より，Schema Graph に対する ObjectRank の重要度を用いることで，上限値をよりタイトに設定することができたことがわかる．

図 5 は DBLP データセットに対する BinRank の事前計算部の実行時間を示している．$maxBinSize$ はクラスタサイズを制御する内部パラメータであり，$maxBinSize$ が大きくなるほどクラスタの数が減少し，索引に格納される部分グラフの数も減少する．実験結果より，BinRank は 10 時間以上もの事前計算を要することが示された．

4.2　近 似 精 度

ObjectRank によって得られた結果を真値とし，提案手法と BinRank の近似精度を計測した．近似精度の指標は以下の式を用いた．

$$Prec(t,k) = \frac{|CompSet(t,k) \cap ORSet(t,k)|}{k}. \quad (18)$$

ただし，$CompSet(t,k)$ はキーワード t に対して提案手法または BinRank を，$ORSet(t,k)$ は ObjectRank をそれぞれ実行した場合に得られた上位 k のノードの集合を表す．

結果を図 6 に示す．実験結果より，提案手法は ObjectRank と全く同一の結果が得られることが示された．提案手法は重要度の上限値と下限値を推定することで上位 k 件になり得ないノードを正確に発見することができるためである．一方，BinRank は事前計算に膨大な時間を要するのにも関わらず，導出される結果は近似解となることが示された．

5　関 連 研 究

PageRank や ObjectRank といったランダムウォークモデルに基づくグラフ分析技術の研究が近年盛んに行われている．既存手法は (1) ノードの枝刈りに基づく手法と (2) 索引を用いた手法の 2 つに分けることができる．

ノードの枝刈りに基づく手法: これらの手法は，グラフから不必要なノードや他のノードの重要度に与える影響が少ないノードを推定し，枝刈りすることで高速化を図る．Sakakura [16] らは，グラフ全体ではなく局所的なグラフを構築することで重要度を推定する手法を提案した．ObjectRank はエッジごとに異なる重みをもち，重みが小さいエッジをもつノードは他のノードへと与える影響も小さい．Sakakura らの手法は，エッジの重みを考慮することで重要度推定に与える影響の小さいノードを推定し，グラフから枝刈りすることで高速な重要度推定を実現した．Fujiwara [7] らは，繰り返し計算の過程で解になり得ないノードを逐次的に枝刈りし，PageRank の Top-k 検索を高速化する手法 F-Rank を提案した．F-Rank では，繰り返し計算の各ステップにおいて各ノードの重要度の上限値と下限値の推定値を用いて解になり得ないノードを特定し，グラフから枝刈りする．グラフからノードの枝刈りを行うが，検索結果は理論的に正しいことが保障されている．しかし，F-Rank は遷移行列の列が正規化される PageRank の特性を利用しているため，ObjectRank には適用できない．

索引を用いた手法: クエリ応答の効率化のために，Chakrabarti ら，Hwang らは索引を用いて高速化を図る手法 HubRank [4]，BinRank [15] をそれぞれ提案した．これらの手法は索引を事前に構築し，クエリ処理時には索引を参照することで検索結果を近似することでクエリ処理時間を短縮する．HubRank は，*hub* ノードと呼ばれるクエリログに基づいて選択された一部のノードの *random walk fingerprints* [6] を事前計算し索引付けする．クエリ処理時には，クエリキーワードを含むノードから幅優先探索を開始し，*hub* ノードに到達するかクエリノードから一定の距離が離れた時点で探索をやめ，得られた部分グラフについ

て索引に基づいた重要度推定を行う．BinRank は，データベースに含まれるキーワードをクラスタリングし，クラスタに含まれるキーワードとの関連度が著しく低いノードを取り除いた部分グラフを，クラスタごとに構築し索引に格納する．クエリ処理時には，クエリキーワードを含むクラスタから作成された部分グラフのみを対象に，べき乗法による重要度評価を行う．

しかし，これらの手法は索引構築に膨大な時間を要するだけでなく，クエリ応答時にも重要度推定のための計算を行う必要があるという問題がある．我々が行った評価実験では，BinRank は文献データベースの索引構築に 10 時間以上かかることが示された（図 5）．

既存手法とは異なり，我々の提案手法は大規模なグラフを対象とした場合でも高速な Top-k 検索が可能である．我々が行った評価実験により，提案手法は事前計算を行わずに従来の ObjectRank と同様の結果を得ることが可能であり，BinRank より高速に結果を得られることが示された．

6 結　　論

本稿では ObjectRank の Top-k 検索の高速化手法を提案した．提案手法では，繰り返し計算の過程で各ノードの重要度の取り得る上限値と下限値を推定することで，上位 k 件になり得ないノードをグラフから逐次的に枝刈りする．この手法により重要度評価の繰り返し計算を進めるにつれてノードが減少するため，従来の ObjectRank と比較して Top-k 検索を高速に行うことができる．また，提案手法はグラフの構造を定義する Schema Graph に対して ObjectRank を実行することで，より高速な枝刈りを実現する．実データを用いた評価実験では，提案手法は従来のべき乗法を用いた ObectRank および既存手法である BinRank と比較して高速に検索結果が得られることを示した．

今後の課題として，複数キーワードによるクエリ応答への対応が挙げられる．従来の ObjectRank では，クエリとして複数のキーワードが与えられた場合，各キーワードについてそれぞれ重要度を計算し，それらの積や和を計算することによって最終的なランキングを決定する．我々の提案手法では，各キーワードについて上位 k 件が完全に独立して導出されるため，それぞれのランキング結果を複合することができないという問題がある．今後は，複数のキーワードからなるクエリに対する，重要度の上限値と下限値を用いた逐次的ノード枝刈り手法の検討を行う．

謝　　辞

本研究の一部は JSPS 科研費 JP18K18057 ならびに JST ACT-I の助成を受けたものである．

文　　献

[1] Bahman Bahmani, Abdur Chowdhury, and Ashish Goel. Fast Incremental and Personalized PageRank. *Proc. VLDB*, Vol. 4, No. 3, pp. 173–184, 2010.

[2] Andrey Balmin, Vagelis Hristidis, and Yannis Papakonstantinou. ObjectRank: Authority-Based Keyword Search in Databases. In *Proc. VLDB*, pp. 564–575, 2004.

[3] Sergey Brin and Lawrence Page. The Anatomy of a Large-Scale Hypertextual Web Search Engine. In *Proc. WWW*, pp. 107–117, 1998.

[4] Soumen Chakrabarti. Dynamic Personalized Pagerank in Entity-Relation Graphs. In *Proc. WWW*, pp. 571–580, 2007.

[5] Paul-Alexandru Chirita, Stefania Costache, Wolfgang Nejdl, and Raluca Paiu. Beagle＋＋: Semantically Enhanced Searching and Ranking on the Desktop. In *Proc. ESWC*, pp. 348–362, 2006.

[6] Dániel Fogaras, Balázs Rácz, Károly Csalogány, and Tamás Sarlós. Towards Scaling Fully Personalized PageRank: Algorithms, Lower Bounds, and Experiments. *Internet Mathematics*, Vol. 2, No. 3, pp. 333–358, 2005.

[7] Yasuhiro Fujiwara, Makoto Nakatsuji, Hiroaki Shiokawa, Takeshi Mishima, and Makoto Onizuka. Fast and Exact Top-k Algorithm for PageRank. In *Proc. AAAI*, pp. 1106–1112, 2013.

[8] Stefania Ghita, Wolfgang Nejdl, and Raluca Paiu. Semantically Rich Recommendations in Social Networks for Sharing, Exchanging and Ranking Semantic Context. In *Proc. ISWC*, pp. 293–307, 2005.

[9] Marco Gori and Augusto Pucci. ItemRank: A Random-Walk Based Scoring Algorithm for Recommender Engines. In *Proc. IJCAI*, pp. 2766–2771, 2007.

[10] Vagelis Hristidis, Heasoo Hwang, and Yannis Papakonstantinou. Authority-Based Keyword Search in Databases. *ACM Trans. Database Syst.*, Vol. 33, No. 1, 2008.

[11] Vagelis Hristidis, Yao Wu, and Louoiqa Raschid. Efficient Ranking on Entity Graphs with Personalized Relationships. *TKDE*, Vol. 26, No. 4, pp. 850–863, April 2014.

[12] Zhipeng Huang and Nikos Mamoulis. Heterogeneous Information Network Embedding for Meta Path based Proximity. *CoRR*, Vol. abs/1701.05291, , 2017.

[13] Zhipeng Huang, Yudian Zheng, Reynold Cheng, Yizhou Sun, Nikos Mamoulis, and Xiang Li. Meta Structure: Computing Relevance in Large Heterogeneous Information Networks. In *Proc. KDD*, pp. 1595–1604, 2016.

[14] Heasoo Hwang, Andrey Balmin, Hamid Pirahesh, and Berthold Reinwald. Information Discovery in Loosely Integrated Data. In *Proc. SIGMOD*, pp. 1147–1149, 2007.

[15] Heasoo Hwang, Andrey Balmin, Berthold Reinwald, and Erik Nijkamp. BinRank: Scaling Dynamic Authority-Based Search Using Materialized SubGraphs. In *Proc. ICDE*, pp. 66–77, 2009.

[16] Yuta Sakakura, Yuto Yamaguchi, Toshiyuki Amagasa, and Hiroyuki Kitagawa. A local method for objectrank estimation. In *Proc. IIWAS*, p. 92. ACM, 2013.

[17] Satu Elisa Schaeffer. Graph Clustering. *Computer Science Review*, Vol. 1, No. 1, pp. 27–64, 2007.

[18] Fabian M Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: a core of semantic knowledge. In *Proc. WWW*, pp. 697–706. ACM, 2007.

[19] Jie Tang, Alvis C. M. Fong, Bo Wang, and Jing Zhang. A Unified Probabilistic Framework for Name Disambiguation in Digital Library. *TKDE*, Vol. 24, No. 6, pp. 975–987, June 2012.

[20] Jie Tang, Limin Yao, Duo Zhang, and Jing Zhang. A Combination Approach to Web User Profiling. *TKDD*, Vol. 5, No. 1, pp. 2:1–2:44, December 2010.

[21] Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. Arnetminer: Extraction and mining of academic social networks. In *Proc. KDD*, 2008.

[22] Yuto Yamaguchi, Tsubasa Takahashi, Toshiyuki Amagasa, and Hiroyuki Kitagawa. TURank: Twitter User Ranking Based on User-Tweet Graph Analysis. In *Proc. WISE*, pp. 240–253, 2010.